**1.What is Object-Oriented Programming, and how does it differ from procedural programming?**

Object-Oriented Programming:

Object-Oriented Programming is programming based on the concept of objects, which are instances of classes. A class is a blueprint or template that defines the characteristics (attributes or properties) and behaviors (methods or functions) that its objects will have.

OOP organizes code around objects and their interactions.

It contains Encapsulation, Inheritance and Polymorphism.

Procedural Programming:

Procedural Programming is a programming paradigm that follows a linear, top-down approach. It is based on procedures or routines (functions) that are executed sequentially. Data and procedures are kept separate.

Procedural programming organizes code around procedures or functions.

It contains Function and Global Data.

**2.Explain the principles of OOP and how they are implemented in Python.**

**Describe the concepts of encapsulation, inheritance, and polymorphism in Python.**

Encapsulation:

Encapsulation is the bundling of data (attributes) and the methods that operate on that data within a single unit or class.

Encapsulation is achieved through the use of classes. Data attributes are defined within a class, and methods that operate on the data are also defined within the same class.

Inheritance:

Inheritance allows a new class (subclass or derived class) to inherit properties and behaviors from an existing class (base class or superclass).

Inheritance is implemented using the syntax class SubClass(BaseClass):. The subclass inherits attributes and methods from the base class. The subclass can also override or extend inherited methods.

Polymorphism:

Polymorphism allows objects of different classes to be treated as objects of a common base class. It enables methods to be used interchangeably for different classes.

Polymorphism is achieved through method overriding. Different classes can provide their own implementation of a method with the same name but specific to their context.

**3.What is the purpose of the self keyword in Python class methods?**

The self is used to represent the instance of the class. Using that keyword, wecan access the attributes and methods of the class in python. It binds the attributes with the given arguments. Python does not use the '@' syntax to refer to instance attributes so we use self keyword.

**4.How does method overriding work in Python, and why is it useful?**

Method overriding in Python refers to the ability of a subclass to provide a specific implementation for a method that is already defined in its superclass. When a method in the subclass has the same name and signature as a method in the superclass, the method in the subclass overrides the method in the superclass.

Method overriding is a key feature of object-oriented programming that enhances code modularity, flexibility, and adaptability. It plays a crucial role in achieving polymorphism

**5.What is the difference between class and instance variables in Python?**

Class Variables:

Class variables are variables that are shared by all instances of a class. They are defined at the class level and are the same for every instance of the class.

Class variables are declared outside of any method within the class and are typically placed near the top of the class definition.

Instance Variables:

Instance variables are variables that belong to a specific instance of a class. Each instance can have its own set of instance variables.

Instance variables are declared within the methods of a class using the self keyword followed by the variable name.

**6.Discuss the concept of abstract classes and how they are implemented in Python.**

Abstract classes are classes in Python that cannot be instantiated on their own and are meant to serve as a blueprint for other classes. They may contain abstract methods, which are declared but not implemented in the abstract class. The responsibility of providing the implementation for these abstract methods falls on the concrete (sub)classes that inherit from the abstract class.

class XYZ(ABC):

@abstractmethod

def area(self):

pass

class add(XYZ):

def \_\_init\_\_(self, len):

self.len = len

def add(self):

return len + len

add = add(len = 10)

**7.Explain the importance of the super() function in Python inheritance.**

The super() function in Python is crucial in the context of inheritance, especially when working with object-oriented programming. It allows a subclass to call methods and access attributes from its superclass, providing a way to extend or override functionality while maintaining a connection to the behavior of the superclass. The super() function is used to invoke methods of the superclass within the subclass and structured approach to inheritance.

**8.How does Python support multiple inheritance, and what challenges can arise from it?**

Python supports multiple inheritance, which means a class can inherit attributes and methods from more than one parent class.

Challenges:

Multiple inheritance can make code more complex and harder to read. Understanding the flow of method resolution and the relationships between classes becomes challenging as the number of parent classes increases.

Multiple inheritance can lead to namespace pollution, where a subclass inherits attributes or methods with the same name from multiple parent classes. This can result in unexpected behavior.

**9.What is a decorator in Python, and how can it be used in the context of OOP?**

Decorator is a special type of function that can be used to modify or extend the behavior of other functions or methods. Decorators provide a clean and concise way to apply additional functionality to functions or methods without modifying their source code.

def deco\_call(func):

def wrapper(self, \*args, \*\*kwargs):

result = func(self, \*args, \*\*kwargs)

return result

return wrapper

class Calculator:

@deco\_call

def add(self, a, b):

return a + b

calculator = Calculator()

result\_add = calculator.add(3, 5)

**10. What do you understand by Descriptive Statistics? Explain by Example.**

Descriptive statistics is a branch of statistics that involves the summary and presentation of data in a meaningful and informative way. It helps in organizing, analyzing, and describing the main features of a dataset, providing insights into the central tendency, variability, and distribution of the data.

It uses measures such as mean, median, mode, range, variance, standard deviation of a dataset.

Example:

25,30,32,35,40,42,45,50,55,60

Mean = 45.4

Median = 42

Mode = no mode

**11. What do you understand by Inferential Statistics? Explain by Example.**

Inferential statistics is a branch of statistics that involves drawing conclusions, making predictions, or making inferences about a population based on a sample of data from that population. It uses probability theory and statistical methods to generalize from a sample to the entire population, making predictions or inferences about population parameters.

Population:

The entire group of interest is the population. In this case, the population is all the employees of the company.

Sample:

A subset of the population from which data is collected is the sample. The company collects income data from 100 randomly selected employees.